# Der 8086/88 als Rechenkünstler

In diesem Referat wird jene Gruppe von Befehlen besprochen, denen der Computer seinen Namen verdankt ("to compute" = engl. rechnen). Dies sind die Arithmetik- und Logikbefehle der 8086/88 - CPU. Die Arithmetikbefehle führen so elementare Operationen wie Addition, Multiplikation oder Division durch, während es die Aufgabe der logischen Befehle ist, zwei Werte nach einer logischen Grundfunktion (UND, ODER, XOR) bitweise zu verknüpfen.

# Inhalt

# 1. Arithmetikbefehle

Fast alle Arithmetikbefehle der 8086/88 - CPU arbeiten mit schlichten 16 - Bit Zahlen. Dabei beschränkt sich der Befehlssatz auf die Grundrechenoperationen. Aus der Sicht des Prozessors "hochkomplizierte" Funktion wie Wurzel oder gar trigonometrische Funktionen gibt es nicht. Das hängt damit zusammen, daß diese Funktionen nur relativ selten benötigt werden. Der größte Teil seiner Arbeit besteht darin, einen Wert von einem Register in den Speicher zu transportieren oder zwei Zahlen zu addieren.

Dabei werden die vorhandenen einfachen Befehle extrem schnell ausgeführt. Rein theoretisch könnten mit einem 8 MHz - CPU in einer Sekunde fast 3,5 Millionen (!) Additionen ausgeführt werden (In der Praxis verlangsamt der Speicherzugriff die Arbeit erheblich).

Die 8086/88 - CPU kann mit zwei verschiedenen Zahlenformaten arbeiten :

Binärformat - Hier wird jedes Bit als Zweierpotenz angesehen. Damit reicht das Format bei vorzeichenlosen Zahlen von 0 bis 255 (8-Bit) bzw. 65535 (16-Bit) und bei vorzeichenbehafteten Zahlen (sogenannte Zweierkomplemetzahlen) von -128 bis +127 bzw. von -32768 bis +32767.

Dezimalformat - Diese sind immer vorzeichenlos. Bei einer Zahl im BCD-Format ("Binary Coded Decimal") wird jede Ziffer separat durch eine 4- oder 8-Bit Zahl dargestellt. Bei einer gepackten Zahl werden in einem Byte zwei Ziffern gespeichert (in jeder Hälfte eine), während im ungepackten Format nur der niederwertige Teil belegt ist, der höherwertige also immer Null enthält. Ein Byte kann also 0 bis 99 darstellen. (siehe 1.8.)

Die Frage ist, wie unterscheidet der Prozessor die beiden Formate ? Antwort : Er tut es gar nicht. Für die beiden Formate sind dieselben Rechenbefehle zuständig. Bei BCD-Zahlen müssen nach der Berechnung spezielle Korrekturbefehle angewandt werden.

## 1.1. Die Additionsbefehle

Die 8086/88 - CPU stellt gleich zwei Befehle für die Addition zur Verfügung, den Befehl ADD (Addiere) und den Befehl ADC (Addiere mit Übertrag). Der Syntax der Befehle ist :

```
<Befehl>  <Zieloperand>, <Quelloperand>
```

Die Befehle addieren den Quelloperanden zum Zieloperanden und legen das Ergebnis im Zieloperanden ab. Folgende Operanden sind zulässig :

| ZIELOPERAND | QUELLOPERAND | BEISPIEL |
|---|---|---|
| <Register> | <Register> | `ADD AX, SI` |
| <Register> | <Speicher> | `ADD BX, X_POS` |
| <Speicher> | <Register> | `ADD Y_POS, CX` |
| <Register> | <Wert> | `ADD AL, 3` |
| <Speicher> | <Wert> | `ADD ANZAHL, 9` |

In dieser Übersicht fällt auf, daß weder eine Addition zweier Speicheroperanden noch die Addition mit einem Segmentregister möglich ist. Die Speicheroperanden bzw. Segmentregister sollten vor einer Addition in Datenregister (AX, BX etc.) geladen werden.

### 1.1.1. ADD

Der ADD - Befehl addiert den Quelloperanden zum Zieloperanden. Die Operanden müssen dabei den gleichen Typ besitzen, also 8- oder 16-Bit. Was passiert nun, wenn zwei 8-Bit Zahlen bei der Addition ein Ergebnis erzeugen, das größer als 8-Bit ist ? In diesem Fall wird durch das Setzen entsprechender Bits ein Übertrag angezeigt. Der sogenannte Carry-Flag wird gesetzt. Um also zu einem gültigen Ergebnis zu gelangen, muß man bei der nächsten Addition einen größeren Typ nehmen. Und außerdem muß man mittels ADC einfach den Carry-Flag als neuntes Bit "dazuaddieren". Entsprechend wird bei einer Addition zwei 16-Bit Zahlen das Carry-Flag gesetzt, wenn das Ergebnis größer als 65535 wird.

Der ADD-Befehl setzt auch noch folgende Flags : Den Vorzeichen-Flag, wenn das Ergebnis negativ ist. Den Null-Flag, wenn das Ergebnis Null wird; sowie den Paritäts-Flag, wenn das Ergebnis gerade ist.

### 1.1.2 ADC

Bei nicht gesetztem Carry-Flag ist der ADC-Befehl mit dem ADD-Befehl identisch. Ansonsten wird zu der normalen Addition noch der Carry-Flag als 9. bzw. 17. Bit dazuaddiert.

## 1.2. Die Subtraktionsbefehle

Die meisten CPU's führen eine Subtraktion durch, indem sie das Zweierkomplement der zu subtrahierenden Zahl, also des Quelloperanden, zum Zieloperanden addieren. Dabei werden negative Ergebnisse, die immer dann auftreten, wenn der Quelloperand größer als der Zieloperand ist, nach wie vor im Zweierkomplement dargestellt. Eine Art Übertrag kann auch bei einer Subtraktion auftreten; dies ist immer dann der Fall, wenn eine größere Zahl von einer kleineren abgezogen wird. Das Carry-Flag wird bei der Subtraktion also immer dann gesetzt, wenn das Ergebnis negativ ist. Außerdem werden gegebenenfalls das Paritäts-Flag, das Null-Flag und das Vorzeichen-Flag gesetzt.

Syntax von SUB und SBB : `<Befehl> <Zieloperand>, <Quelloperand>`

Unmittelbaren Werte dürfen nicht eingesetzt werden, nur Register und Speicheradressen.

### 1.2.1. SUB

Es kann passieren, daß bei einer Subtraktion zweier Zahlen mit unterschiedlichem Vorzeichen ein Ergebnis entsteht, das nicht mehr im Darstellungsbereich einer Zweierkomplementzahl liegt. Wie bei allen arithmetischen Befehlen üblich, wird in diesem Fall das Überlauf-Flag gesetzt. Beispiel : Die 8-Bit Zahl 40 wird von der 8-Bit Zahl -100 abgezogen. Natürlich hat die Zahl -140 in dem 8-Bit Ergebnis keinen Platz. Dies wird durch den gesetzten Überlauf-Flag angezeigt. Um ein gültiges Ergebnis zu erhalten, müssen vor der Subtraktion beide Operanden auf 16-Bit erweitert werden.

### 1.2.2. SBB

Dieser Befehl entspricht genau dem SUB - Befehl. Nur wird hier zusätzlich das Carry-Flag vom Zieloperanden abgezogen. Auf diese Weise können, analog zur Addition, die höherwertigen Hälften von Zahlen mit mehr als 16-Bit subtrahiert werden.

## 1.3. Die Multiplikationsbefehle

Die 8086/88 - CPU gehörte seinerzeit (1978) zu den ersten CPU's, die über einen Multiplikations- und einen Divisionsbefehl verfügten. Ohne einen solchen Befehl müssen derartige Aufgaben durch eine Folge von Additions- und Subtraktionsbefehlen ersetzt werden, was natürlich ziemlich langsam ist und den Programmcode vergrößerte. Bei der 8086/88 - CPU gibt es die Befehle MUL und IMUL.

Syntax : `<Befehl> <Quelloperand>`

Bei dem Quelloperanden handelt es sich entweder um ein Register oder eine Speichervariable. Der Zieloperand wird immer in AL- bzw. im AX- Register erwartet. Werden zwei 8-Bit Zahlen miteinander multipliziert, wird der Zieloperand aus dem AL - Register geholt; werden dagegen 16-Bit Zahlen multipliziert, kommt der Zieloperand aus dem AX-Register. Aber wohin kommt das Ergebnis ? Bei einer 8-Bit (Byte-) Multiplikation wird das 16-Bit Ergebnis im AX - Register abgelegt, wobei das AH - Register die höherwertigen und das AL - Register die niederwertigen 8 Bit enthält. Bei einer Multiplikation von zwei 16-Bit Zahlen entsteht ein 32-Bit Ergebnis. Da hierfür ein einzelnes CPU - Register nicht mehr ausreicht, werden die niederwertigen 16 Bit im AX - Register und die höherwertigen 16 Bit im DX - Register gespeichert.

Die Multiplikationsbefehle beeinflussen die folgenden Flags : Der Carry-Flag wird in Verbindung mit dem Überlauf-Flag gesetzt, wenn das Ergebnis größer als das darstellbare Format des Zieloperanden liegt. Außerdem werden gegebenenfalls das Paritäts-Flag, das Null-Flag; sowie das Vorzeichen-Flag gesetzt.

### 1.3.1. MUL

Der MUL - Befehl ist für vorzeichenlose Zahlen zuständig. Wenn man bei MUL- oder IMUL - Befehlen als Quelloperanden AL bzw. AX angibt, erhält man praktisch die Quadratfunktion. Zu beachten ist bei beiden Befehlen außerdem, daß eine Multiplikation mit einem unmittelbaren Wert nicht erlaubt ist (MUL 2 o.ä.).

### 1.3.2. IMUL

Dieser Befehl ist für die Multiplikation zweier vorzeichenbehafteten Zahlen zuständig.

## 1.4. Die Divisionsbefehle

Auch hier stellt die 8086/88 - CPU zwei Befehle zur Verfügung : der DIV- und IDIV-Befehl. Bei der Division wird, wie bei der Multiplikation, der Quelloperand durch den Zieloperand geteilt. Da die Divisionsbefehle mit ganzzahligen Integer - Zahlen arbeiten, kann das Ergebnis auch nur ganzzahlig sein. Man erhält bei der Division also stets ein ganzzahliges Ergebnis und einen ganzzahligen Rest. Mit Hilfe des Restes kann durch fortlaufende Division jedoch eine beliebige Genauigkeit, auch ohne Fließkommaarithmetik, erreicht werden. Bei einer 16-Bit Division wird das 8-Bit Ergebnis in AL und der 8-Bit Rest in AH abgelegt. Bei einer 32-Bit Division wird analog das 16-Bit Ergebnis in AX, der 16-Bit Rest in DX abgelegt.

Bei der Verwendung des DIV - Befehls wird oft vergessen, daß bei einer 32-Bit Division immer auch das DX - Register beteiligt ist, auch wenn eine Zahl dividiert werden soll, die nicht die vollen 32 Bit ausnutzt. Der Grund für einen scheinbar unbegründeten

Divisionsüberlauf ist oft, das vergessen wurde, das DX - Register vor der Division auf Null zu setzen.

Eine weitere Komplikation kann bei der Division auftreten : Der Divisons-Überlauf ("Divide Overflow"). Dieser kann entweder auftreten, wenn das Ergebnis oder der Rest nicht in sein Register paßt, oder bei der Division durch Null. In diesem Fall wird kein Flag gesetzt, sondern der Interrupt 0 der 8086/88 - CPU ausgeführt. Im Verlauf dieses Interrupts wird der gerade bearbeitete Divisionsbefehl abgebrochen, und der Programmlauf an der nächsten Anweisung im Programm fortgesetzt. Dabei gehen die Inhalte der Register AX und DX verloren, sie enthalten nur noch "Datenschrott". Falls das Programm den Interrupt Null unverändert läßt, wird durch ein Divisionsüberlauf das Programm mit der Meldung "Überlauf bei Division" abgebrochen.

Laut Datenblatt der 8086/88 - CPU ist der Zustand der Status-Flags nach einer Division nicht definiert. Dies wird nicht umsonst so sein, man sollte also keine Prüfung vornehmen, da die Belegung sich bei verschiedenen Prozessorgenerationen ändern kann. Eine Prüfung bzw. Auswertung der Zustände kann gut gehen, muß aber nicht.

### 1.4.1. DIV

Der DIV - Befehl behandelt vorzeichenlose Zahlen. Der Syntax lautet :

```
DIV  <Quelloperand>
```

Bei einer 16-Bit Divison wird also zum Beispiel mit `DIV BL` aufgerufen. Hier ist der Divisior 8-Bit, also ist der Dividend 16-Bit, und kommt aus AX. Das Ergebnis wird dann in AL, der Rest in AH abgelegt. Bei einer 32-Bit Division dagegen, zum Beispiel `DIV BX`, kommt der Dividend aus dem Registerpaar AX:DX. Das Ergebnis wird in diesem Fall in AX, der Rest in DX abgelegt.

### 1.4.2. IDIV

Dieser hat denselben Syntax wie der DIV - Befehl, also `IDIV <Quelloperand>`. Genau wie beim DIV - Befehl und den Multiplikationsbefehlen darf auch hier kein unmittelbarer Wert angegeben werden, sondern nur ein Register oder eine Speicheradresse. Das Ergebnis muß im darstellbaren Bereich für vorzeichenbehaftete Zahlen liegen, sonst kommt es zu einem Divisionsüberlauf.

## 1.5. Die Befehle INC und DEC

Diese Befehle sind sehr häufig vorkommende Rechenoperationen; sie werden hauptsächlich zur Schleifenzählung eingesetzt. Dabei ist allerdings zu beachten, daß weder der INC-, noch der DEC - Befehl das Carry-Flag beeinflussen. Dieses darf also nicht zur Prüfung der Abbruchbedingung herangezogen werden (z.B. durch JNC - Befehl).

Syntax :

| | | |
|---|---|---|
| `INC` | `<Zieloperand>` | erhöht den Operanden um eins. |
| `DEC` | `<Zieloperand>` | erniedrigt den Operanden um eins. |

## 1.6. Der NEG - Befehl

Syntax : `NEG <Zieloperand>.` Dieser Befehl negiert den Operanden, indem er ihn von Null abzieht. Da negative Zahlen stets im Zweierkomplement dargestellt werden, bietet dieser Befehl eine elegante Möglichkeit zur Bildung des Zweierkomplements eines Operanden. Eine weitere häufige Anwendung des Befehls ist die Subtraktion eines Operanden von einem unmittelbaren Wert, da der SUB - Befehl dies nicht zuläßt. Stattdessen führt man die Subtraktion `SUB <Wert>, AL` z.B. so aus :

```
NEG AL
ADD AL, <Wert>
```

Der NEG - Befehl beeinflußt auch die Status-Flags. So wird das Carry-Flag und das Vorzeichen-Flag gesetzt, wenn das Ergebnis negativ ist. Das Null-Flag wird gesetzt, wenn das Ergebnis Null ist; der Paritäts-Flag, wenn das Ergebnis gerade ist; und das Überlauf-Flag, wenn das Ergebnis nicht dargestellt werden kann (zum Beispiel, wenn bei einem 8-Bit Typ der Wert -128 negiert werden soll).

## 1.7. Die Befehle zur Vorzeichenerweiterung

Aus den bisherigen Beispielen, wo die zulässigen Typen für Operanden besprochen wurden, wurde deutlich, daß bei allen arithmetischen Operationen beide Operanden die gleiche Größe besitzen müssen (vorzeichenlos oder -behaftet). Zur Erweiterung von vorzeichenlosen Operanden auf vorzeichenbehaftetes Format gibt es die Befehle CBW ("Convert Byte to Word") und CWD ("Convert Word to Doubleword"). Es werden für diese Befehle keine Operanden benötigt, vielmehr erwarten diese ihre Werte in den Registern AL (bei CBW) bzw. AX (CWD). Das erweiterte Ergebnis wird danach in AX (bei CBW) bzw. AX:DX (CWD) abgelegt. Syntax für CBW und CWD : `<Befehl>`.

## 1.8. Rechnen mit BCD - Zahlen

Da der Prozessor keine speziellen Befehle für die Bearbeitung von BCD - Zahlen kennt, entstehen bei der Anwendung der - eigentlich für Binärzahlen gedachten - Befehle zwangsläufig Fehler. Nach einer arithmetischen Operation, die auf zwei BCD - Zahlen angewandt wird, ist daher eine Korrektur notwendig. Dafür gibt es entsprechende Befehle der 8086/88 - CPU.

Da es allerdings zwei Unterformate der BCD - Zahlen gibt, nämlich das gepackte und das ungepackte Format, müssen auch zwei verschiedene Korrekturbefehle für diese Unterformate vorhanden sein. Die Bezeichnungen "ASCII - Korrektur" und "Dezimal - Korrektur" sollen hier erklärt werden : Da eine ungepackte BCD - Zahl durch Addition von Dezimal 48 in den entsprechenden ASCII - Code umgewandelt werden kann, wird sie auch als "ASCII - Dezimalzahl" bezeichnet. Das gepackte Format dagegen wird in diesem Zusammenhang als Dezimalzahl bezeichnet. Daher die Anfangsbuchstaben der Korrekturbefehle : "A" für "ASCII" bzw. "D" für "Dezimal".

Der Syntax aller nun folgenden Befehle ist gleich. Alle erwarten ihren Operanden im Register AX. Bei Berechnungen mit BCD - Zahlen sollte bei den vorangehenden Arithmetikbefehlen als Zieloperand also immer AX angegeben werden. Syntax für die Befehle AAA, DAA, AAS, DAS, AAM, AAD ist also : `<Befehl>`.

### 1.8.1. Korrektur nach Addition : AAA, DAA

Das Ergebnis des AAA - Befehls ("ASCII Adjust after Addition") ist eine ungepackte BCD - Zahl, im Gegensatz zum DAA - Befehl ("Decimal Adjust AL after Addition"), der eine gepackte BCD - Zahl erzeugt. Beide Befehle erreichen dies durch die Addition von "6" zum AL - Wert, und, falls hierbei ein Übertrag entsteht, die Erhöhung des AH - Registers um "1".

### 1.8.2. Korrektur nach Subtraktion : AAS, DAS

Das Vorgehen der Befehle AAS ("ASCII Adjust AL after Subtraction") und DAS ("Decimal Adjust AL after Subtraction") ist ähnlich : Falls das AL - Register eine Zahl größer als neun enthält, so ziehen diese Befehle sechs vom AL - Register ab und erniedrigen AH um eins. In beiden Fällen werden die beiden höherwertigen Bits im AL - Register gelöscht, so daß eine 4-Bit Zahl übrigbleibt. Beim DAS - Befehl wird zusätzlich der Inhalt des AH - Registers in eine gültige BCD - Ziffer umgewandelt.

### 1.8.3. Korrektur nach Multiplikation : AAM

Zu Beachten ist, daß die 8086/88 - CPU keine Korrektur für gepackte Zahlen zur Verfügung stellt. Vor einer Multiplikation muß AL also in zwei ungepackte Zahlen umgewandelt werden (z.B. Bit 4 bis 7 von AL in Bit 0 bis 3 in BL kopieren; dann Bits 4 bis 7 in AL und in BL gleich Null setzen). Der AAM - Befehl ("ASCII Adjust AX after Multiply") wandelt das Produkt einer Byte - Multiplikation in zwei ungepackte BCD - Zahlen um, die im Register AX abgelegt werden.

### 1.8.4. Korrektur für Division : AAD

Der AAD - Befehl ("ASCII Adjust before Devision") wird, im Gegensatz zu den übrigen Konvertierungsbefehlen, vor der Division auf die Operanden angewandt. Dazu wird vom AAD - Befehl ein zweistelliger ungepackter Divisor im BCD - Format in das Binärformat umgewandelt. Mit dieser Zahl kann dann die Division erfolgen.

# 2. Logische Verknüpfungen

Die 8086/88 - CPU baut auf den logischen Grundverknüpfungen UND, ODER und EXOR auf. Die logischen Befehle AND, OR und XOR arbeiten mit zwei Operanden.

Deren Syntax ist also : `<Befehl> <Zieloperand>, <Quelloperand>`
Der Syntax des NOT - Befehls dagegen ist : `NOT <Zieloperand>`

Alle Bits des Quelloperanden werden der Reihe nach einzeln mit den entsprechenden Bits im Zieloperanden, nach der Wahrheitstabelle miteinander verknüpft. Das Ergebnis wird im Zieloperanden abgelegt.

In der Praxis werden logische Befehle sehr häufig zum Setzen und Rücksetzen einzelner Bits benutzt. Dieses Manipulieren einzelner Bits bezeichnet man auch als "Maskieren".

Die Status-Flags werden von den logischen Verknüpfungen beeinflußt : So werden das Überlauf- und das Carry-Flag stets auf Null gesetzt, da sie keine Bedeutung haben können. Das Vorzeichen-, das Null- und das Paritäts-Flag werden in Abhängigkeit von dem Ergebnis der logischen Verknüpfung gesetzt.

## 2.1. Die AND, OR, XOR und NOT - Befehle

Folgende Grundverknüpfungen werden ausgeführt :
AND Logische UND - Verknüpfung
OR Logische ODER - Verknüpfung
XOR Logische EXOR - Verknüpfung
NOT Invertierung (Bilden des Einerkomplements)

Mögliche Anwendungen der logischen Befehle :

| ZIELOPERAND | QUELLOPERAND | BEISPIEL |
|---|---|---|
| <Register> | <Register> | `AND AX, CX` |
| <Register> | <Speicher> | `OR SI, ZEIGER1` |
| <Register> | <Wert> | `AND AH, 7Fh` |
| <Register> | <Wert> | `AND AL, 10101100b` |
| <Speicher> | <Register> | `XOR MUSTER, CL` |
| <Speicher> | <Wert> | `OR VIDEO, 8000h` |

Interessant ist das vierte Beispiel : Hier wurde direkt eine Binärzahl angegeben. Das erhöht die Übersichtlichkeit enorm, speziell bei einer Maskierung einzelner Bits. Hier wird sofort deutlich, welchen Effekt die logische Verknüpfung hat.

## 2.2. Der TEST - Befehl

Der TEST - Befehl ist sehr ähnlich dem AND - Befehl. Der einzige Unterschied besteht darin, daß nicht der Zieloperand verändert wird, sondern lediglich die Status-Flags entsprechend des Ergebnisses gesetzt werden. Mit dem TEST - Befehl kann also der Zustand einzelner Bits in einem Operanden testen, ohne den Operanden zu verändern. Mit dem Ergebnis, den gesetzten Flags, können anschließend bedingte Sprünge (wie zum Beispiel die Befehle JNC, JZ oder ähnliche) ausgeführt werden.

# 3. Schiebe- und Rotationsbefehle

Die Schiebe- und Rotationsbefehle sind eine Art "Zwischending" zwischen den Arithmetik- und den Logik - Befehlen dar. Zum einen lassen sich mit ihnen arithmetische Grundoperationen wie zum Beispiel Multiplikation und Division berechnen; der Zieloperand kann aber trotzdem auch als Ansammlung einzelner Bits und nicht als Zahlenwert betrachtet werden.

Der Syntax aller Schiebe- und Rotationsbefehle lautet :

```
<Befehl>  <Zieloperand>, <Quelloperand>
```

Als Zieloperand sind Register und Speicher zulässig, als Quelloperand entweder der unmittelbare Wert eins (1) oder das CL - Register. Die Anzahl der Verschiebungen oder Rotationen ist also entweder 1 oder wird durch den Inhalt des CL - Registers angegeben.

## 3.1. Die Schiebebefehle

Bei den Schiebebefehlen gibt es zwei Varianten : Man unterscheidet zwischen einer arithmetischen und einer logischen Verschiebung. Der Unterschied liegt darin, daß bei einer logischen Verschiebung das Vorzeichen nicht beachtet wird, im Gegensatz zu der arithmetischen Verschiebung, wo "durch das Vorzeichen hindurch" verschoben wird.

Bei einer Verschiebung muß man sich den Inhalt des Operanden als eine Kette von 8 oder 16 Bits vorstellen. Durch einen entsprechenden Befehl wird die Kette nach links oder rechts verschoben. Dabei fällt am Ende ein Bit heraus (wird in das Carry-Flag abgelegt), während am Anfang eine Null hineingeschoben wird. Die einzelnen Schiebebefehle unterscheiden sich dadurch, wie das freigewordene Bit besetzt wird. Bei der logischen Rechtsverschiebung wird die Null in das höchstwertige Bit eingeschoben, bei der arithmetischen dagegen nicht, da hier das Vorzeichen ist und erhalten bleiben soll. Bei beiden Linksverschiebungen, also sowohl der logischen als auch der arithmetischen, rückt eine Null in die freigewordene Bit - Position Null.

### 3.1.1. SAL

"Shift Arithmetic Left" - Identisch mit SHL, wird deshalb von manchen DEBUG - Versionen nicht akzeptiert. Rückt alle Bits um <Quelloperand> Schritte nach links, alle freigewordene Bit-Positionen werden Null. Letztes "rausgeschmissenes" Bit wird im Carry-Flag abgelegt.

### 3.1.2. SAR

"Shift Arithmetic Right" - Rückt bis auf höchstwertiges Bit alle Bit-Position um <Quell-operand> Schritte nach rechts. Carry-Flag wird vom letzten "verlorenen" Bit belegt.

### 3.1.3. SHL

"Shift Logical Left" - Ist mit dem SAL - Befehl identisch. Siehe dort.

### 3.1.4. SHR

"Shift Logical Right" - Rückt alle Bits einschließlich dem höchstwertigen um <Quell-operand> Schritte nach rechts. Das letzte "verlorene" Bit wird im Carry-Flag gespeichert.

## 3.2. Die Rotationsbefehle

Auch die Rotationsbefehle verschieben, ähnlich den Schiebebefehlen, einen Operanden um eine oder mehrere Positionen. Hier geht allerdings kein Bit verloren, sondern das am einen Ende herausgeschobene Bit wird am anderen Ende wieder hineingeschoben. Dabei unterscheidet man zwischen zwei Methoden. Bei den Befehlen ROL und ROR wird ohne den Carry-Flag rotiert, bei den Befehlen RCL und RCR wird zusätzlich "durch das Carry-Flag hindurch" rotiert. Um bei der Rotation mit dem Carry-Flag das gewünschte Ergebnis zu erhalten, stellt die 8086/88 - CPU drei Befehle zur Verfügung, mit denen man den Zustand des Carry-Flags beeinflussen kann. Diese sind :

| | | |
|---|---|---|
| `STC` | Setzt das Carry-Flag auf "1" | ("Set Carry-Flag") |
| `CLC` | Setzt das Carry-Flag auf "0" | ("Clear Carry-Flag") |
| `CMC` | Invertiert den Zustand des Carry-Flags | |

### 3.2.1. ROL

("ROtate Left") - Rotiert alle Bit-Positionen des <Zieloperanden> um <Quelloperand> Schritte nach links, wobei "hinausgeschobene" Bits rechts wieder eingeschoben werden und das letzte hinausgeschobene Bit zusätzlich im Carry-Flag gespeichert wird.

### 3.2.2. ROR

("ROtate Right") - Rotiert alle Bit-Positionen des <Zieloperanden> um <Quelloperand> Schritte nach rechts, wobei "hinausgeschobene" Bits links wieder eingeschoben werden und das letzte hinausgeschobene Bit zusätzlich im Carry-Flag ablegt wird. Praktisch lassen sich mit dem ROR - Befehl die niederwertigen vier Bits eines 8-Bit Operanden mit den höherwertigen vier Bits vertauschen.

### 3.2.3. RCL

("Rotate through Carry Left") - Hier wird das Carry-Flag in der Rotation miteinbezogen. Es kann also quasi als 9. bzw. als 17. Bit in einem 9- bzw. 17-Bit Register arbeiten.

### 3.2.4. RCR

("Rotate through Carry Right") - Hier wird der Carry-Flag in die Rotation miteinbezogen.